Operating System and Design (19CS2106S)
Lab- 7

Pre-Lab:

The brk and sbrk calls dynamically change the amount of space allocated for the data segment of the calling process. standard library functions: malloc(), calloc(), free() and realloc(). C memory functions: memcpy, memmove, memcmp, memchr, memset. C string functions .

OSD - practical - 7 190031187
Radhakrishna

pre-lab

1. standard library functions :-

malloc ()

malloc is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form. It intializes each block with default garbage value.

syntax :- (cast-type *) malloc (byte-size);

ptr = (int *) malloc (10 * sizeof(int));

Calloc():-

calloc is used to dynamically allocate the specified no. of blocks of memory of the specified type. It initializes each block with a default value '0'.

syntax ptr = (cast-type *) calloc (n, element-size);

free():-

free is used to dynamically de-allocate the memory. Memory allocated using malloc(), calloc() is not de-allocated on their own. Hence, the free() is used, whenever

the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

Syntax: free(ptr);

realloc():

realloc is used to dynamically change the memory allocation of a previously allocated memory.

Syntax:

ptr = realloc(ptr, newSize);

where ptr is reallocated with new size "newSize"

C memory functions :-

memcpy()

memcpy() is used to copy a block of memory from a location to another. It is declared in string.h.

Syntax:

void *memcpy(void *to, const void *from, size_t numBytes)

memmove()

memmove() is another library function that handles overlapping well.

memcmp()

It compares the first count characters of the arrays pointed to by buf1 and buf2

Syntax:-

```
int memcmp(const void *buf1, const void *buf2
                size_t count)
```

memchr()

memchr() is a function which will search for the first occurance of a character in a specified no. of characters.

Syntax:-

```
const void * memchr(const void *ptr,
        int ch, std::size_t count);
```

memset()

memset() is used to fill a block of memory with a particular value.

Syntax:-

```
void *memset (void *ptr, int x, size_t n);
```

osd-190031187@team-osd:~

```
  GNU nano 2.3.1              File: prelab7.c                          Modified

#include<stdio.h>
int main(void)
{
 static int data;
 return 0;
}
```

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ nano prelab7.c
[osd-190031187@team-osd ~]$ gcc prelab7.c -o prelab7
[osd-190031187@team-osd ~]$ size prelab7
   text    data     bss     dec     hex filename
   1127     540      12    1679     68f prelab7
[osd-190031187@team-osd ~]$
```

osd-190031187@team-osd:~

```
  GNU nano 2.3.1              File: prelab7.c                          Modified

#include<stdio.h>
int main(void)
{
 static int data=10;
 return 0;
}
```

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ nano prelab7.c
[osd-190031187@team-osd ~]$ gcc prelab7.c -o prelab7
[osd-190031187@team-osd ~]$ size prelab7
   text    data     bss     dec     hex filename
   1127     544       8    1679     68f prelab7
[osd-190031187@team-osd ~]$
```

osd-190031187@team-osd:~

```
  GNU nano 2.3.1              File: prelab7.c                          Modified

#include<stdio.h>
int main(void)
{

 return 0;
}
```

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ nano prelab7.c
[osd-190031187@team-osd ~]$ gcc prelab7.c -o prelab7
[osd-190031187@team-osd ~]$ size prelab7
   text    data     bss     dec     hex filename
   1127     540       4    1671     687 prelab7
[osd-190031187@team-osd ~]$
```

osd-190031187@team-osd:~

```
  GNU nano 2.3.1              File: prelab7.c

#include<stdio.h>
int main(void)
{
 static int data2;
 return 0;
}
```

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ nano prelab7.c
[osd-190031187@team-osd ~]$ gcc prelab7.c -o prelab7
[osd-190031187@team-osd ~]$ size prelab7
   text    data     bss     dec     hex filename
   1127     540      12    1679     68f prelab7
[osd-190031187@team-osd ~]$
```

osd-190031187@team-osd:~

```
  GNU nano 2.3.1              File: prelab7.c                    Modified

#include<stdio.h>
int main(void)
{
 static int data2=20;
 return 0;
}
```

osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ nano prelab7.c
[osd-190031187@team-osd ~]$ gcc prelab7.c -o prelab7
[osd-190031187@team-osd ~]$ size prelab7
   text    data     bss     dec     hex filename
   1127     544       8    1679     68f prelab7
[osd-190031187@team-osd ~]$
```

In-Lab

1. Write a program to display the address space of various segments (stack, heap, data ...etc) and show that memory address a programmer see is virtual not real.

CODE

```
#include<stdio.h>
#include<malloc.h>
int glb_uninit; /* Part of BSS Segment -- global uninitialized
variable, at runtime it is
initialized to zero */
int glb_init = 10;
/* Part of DATA Segment -- global initialized variable */
void foo(void)
{
static int num = 0;
/* stack frame count */
int autovar;
/* automatic variable/Local variable */
int *ptr_foo = (int*)malloc(sizeof(int));
if (++num == 4)
/* Creating four stack frames */
return;
printf("Stack frame number %d: address of autovar: %p\n", num, &
autovar);
printf("Address of heap allocated inside foo() %p\n",ptr_foo);
foo();
/* function call */
}
int main()
{
char *p, *b, *nb;
int *ptr_main = (int*)malloc(sizeof(int));
printf("Text Segment:\n");
printf("Address of main: %p\n", main);
printf("Address of afunc: %p\n",foo);
printf("Stack Locations:\n");
foo();
printf("Data Segment:\n");
printf("Address of glb_init: %p\n", & glb_init);
printf("BSS Segment:\n");
printf("Address of glb_uninit: %p\n", & glb_uninit);
printf("Heap Segment:\n");
printf("Address of heap allocated inside main() %p\n",ptr_main);
return 0;
}
```

OUTPUT

```
osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ nano lab7_inlab1.c
[osd-190031187@team-osd ~]$ gcc lab7_inlab1.c
[osd-190031187@team-osd ~]$ ./a.out
Text segment:
Address of main: 0x400625
Address of afunc: 0x4005bd
Stack Locations:
Stack frame number 1: address of autovar: 0x7ffce73d7f84
Address of heap allocated inside foo() 0x24e3030
Stack frame number 2: address of autovar: 0x7ffce73d7f64
Address of heap allocated inside foo() 0x24e3050
Stack frame number 3: address of autovar: 0x7ffce73d7f44
Address of heap allocated inside foo() 0x24e3070
Data Segment:
Address of glb_init: 0x601044
BSS Segment:
Address of glb_uninit: 0x601050
Heap Segment:
Address of heap allocated inside main() 0x24e3010
[osd-190031187@team-osd ~]$
```

2. Develop a program to illustrate the effect of free() on the program break. This program allocates multiple blocks of memory and then frees some or all of them, depending on its (optional) command-line arguments.

CODE

```
#define MAX_ALLOCS 1000000
#include <stdio.h> /* Standard I/O functions */
#include <stdlib.h> /* Prototypes of commonly used library
functions,plus EXIT_SUCCESS and EXIT_FAILURE constants */
#include <unistd.h> /* Prototypes for many system calls */
#include <errno.h> /* Declares errno and defines error constants */
#include <string.h> /* Commonly used string-handling functions */
int main(int argc, char *argv[]){
char *ptr[MAX_ALLOCS];
int freeStep, freeMin, freeMax, blockSize, numAllocs,j;
printf("\n");
if (argc < 3 || strcmp(argv[1], "--help") == 0){
printf("%s num-allocs block-size [step [min [max]]]\n" argv[0]);
exit(5); }
numAllocs = strtol(argv[1], NULL, 10);
if (numAllocs > MAX_ALLOCS){
printf("num-allocs > %d\n", MAX_ALLOCS);
exit(5); }
blockSize = strtol(argv[2], NULL, 10);
freeStep = (argc > 3) ? strtol(argv[3], NULL, 10): 1;
freeMin = (argc > 4) ? strtol(argv[4], NULL, 10) : 1;
freeMax = (argc > 5) ? strtol(argv[5], NULL, 10) : numAllocs;
if (freeMax > numAllocs){ printf("free-max > num-allocs\n");
exit(5); }
printf("Initial program break: %10p\n", sbrk(0));
printf("Allocating %d*%d bytes\n", numAllocs, blockSize);
for (j = 0; j < numAllocs; j++) {ptr[j] = malloc(blockSize);
```

```
if (ptr[j] == NULL){ perror("malloc");
exit(5); }}
printf("Program break is now: %10p\n", sbrk(0));
printf("Freeing blocks from %d to %d in steps of %d\n",freeMin,
freeMax, freeStep);
for (j = freeMin -1;
j < freeMax;
j += freeStep)free(ptr[j]);
printf("After free(), program break is: %10p\n", sbrk(0));
exit(10);
```

OUTPUT

```
osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ nano lab7_inlab2.c
[osd-190031187@team-osd ~]$ gcc lab7_inlab2.c
[osd-190031187@team-osd ~]$ ./a.out 1000 10240 2

Initial program break:  0x1c73000
Allocating 1000*10240 bytes
Program break is now:  0x263b000
Freeing blocks from 1 to 1000 in steps of 2
After free(), program break is:  0x263b000
[osd-190031187@team-osd ~]$ ./a.out 1000 10240 1 1 999

Initial program break:  0x1bf9000
Allocating 1000*10240 bytes
Program break is now:  0x25c1000
Freeing blocks from 1 to 999 in steps of 1
After free(), program break is:  0x25c1000
[osd-190031187@team-osd ~]$ ./a.out 1000 10240 1 500 1000

Initial program break:  0x22c6000
Allocating 1000*10240 bytes
Program break is now:  0x2c8e000
Freeing blocks from 500 to 1000 in steps of 1
After free(), program break is:  0x27c8000
[osd-190031187@team-osd ~]$
```

Post-Lab

1. Write a simple memory allocator: memalloc is a simple memory allocator. Which uses your own malloc(), calloc(), realloc() and free() implemented using system calls.

CODE

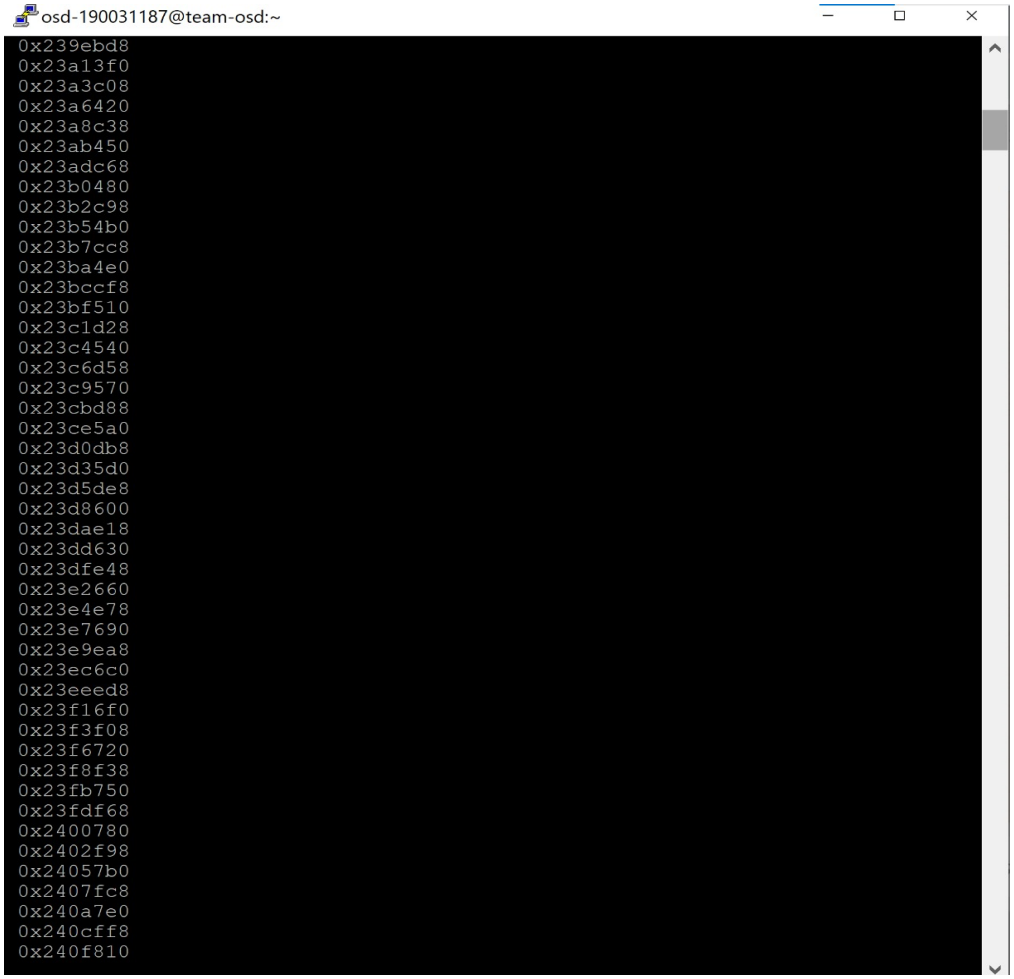
```
#include <sys/types.h> /* Type definitions used by many programs */

#include <stdio.h> /* Standard I/O functions */

#include <stdlib.h> /* Prototypes of commonly used library
functions,plus EXIT_SUCCESS and EXIT_FAILURE constants */

#include <unistd.h> /* Prototypes for many system calls */

#include <errno.h> /* Declares errno and defines error constants */

#include <string.h> /* Commonly used string-handling functions */

extern char end;

void *my_malloc (size_t);

void my_free(void *);

struct blk {size_t size;

struct blk *prev;

struct blk *next;};

struct blk *first = NULL;

struct blk *last = NULL;

void *my_malloc (size_t size) {size_t required_size = size +
sizeof(struct blk);

struct blk *curr = first;

while (curr != NULL && curr->size < required_size) {curr = curr-
>next;

}if (curr == NULL) {void *new = sbrk((intptr_t) required_size);

if (new == (void *) -1) { return NULL; }

struct blk *new_blk = (struct blk *) new;

new_blk->size = required_size;
```

```
return (void *) (new_blk + 1);}

if (curr == first) { first = first->next; }

else { curr->prev->next = curr->next; }

if (curr == last) { last = last->prev; }

else {curr->next->prev = curr->prev; }if (curr->size > required_size
+ sizeof(struct blk)) {struct blk *left = (struct blk *) (((char *)
curr) + required_size);

left->size = curr->size -required_size;

curr->size = required_size;

my_free((char *) (left + 1));}return (void *) (curr + 1);}

void my_free (void *ptr) {struct blk *blk_ptr = ((struct blk *) ptr)
-1;

if (first == NULL) {first = last = blk_ptr;return;}if (blk_ptr <
first) {blk_ptr->prev = NULL;

if (((char *) blk_ptr) + blk_ptr->size == (char *) first) {blk_ptr-
>size += first->size;

blk_ptr->next = first->next;}

 else {first->prev = blk_ptr;blk_ptr->next = first;}first =
blk_ptr;return;}

if (blk_ptr > last) {if (((char *) last) + last->size == (char *)
blk_ptr) {last->size += blk_ptr->size;}

else {blk_ptr->next = NULL;

blk_ptr->prev = last;

last->next = blk_ptr;

last = blk_ptr;}

return;}

struct blk *curr = first;

while (curr < blk_ptr) {curr = curr->next;}

struct blk *before = curr->prev;

if (((char *) before) + before->size == (char *) blk_ptr) {before-
>size += blk_ptr->size;

blk_ptr = before;}
```

```
 else {blk_ptr->prev = before;

before->next = blk_ptr;}

if (((char *) blk_ptr) + blk_ptr->size == (char *) curr) {blk_ptr-
>size += curr->size;

blk_ptr->next = curr->next;

curr->next->prev = blk_ptr;




} else {blk_ptr->next = curr;

curr->prev = blk_ptr;}}

#define MAX_ALLOCS 1000000

int main (int argc, char *argv[]) {

/* copied from free_and_sbrk.c --licensed by Michael Kerrisk under
the GPLv3 */

char *ptr[MAX_ALLOCS];

int freeStep, freeMin, freeMax, blockSize, numAllocs, j;

printf("\n");

if (argc < 3 || strcmp(argv[1], "--help") == 0) {printf("%s num-
allocs block-size [step [min [max]]]\n", argv[0]);

perror("num-allocs block-size");}

numAllocs = strtol(argv[1], NULL, 10);

if (numAllocs > MAX_ALLOCS) {printf("num-allocs > %d\n",
MAX_ALLOCS);

perror("num-allocs");}

blockSize = strtol(argv[2], NULL, 10);

freeStep = (argc > 3) ? strtol(argv[3], NULL, 10) : 1;

freeMin = (argc > 4) ? strtol(argv[4], NULL, 10) : 1;

freeMax = (argc > 5) ? strtol(argv[5], NULL, 10): numAllocs;

if (freeMax > numAllocs) {perror("free-max > num-allocs");}

printf("Initial program break: %10p\n", sbrk(0));
```

```
printf("Allocating %d*%d bytes\n", numAllocs, blockSize);

for (j = 0; j < numAllocs; j++) {

ptr[j] = my_malloc(blockSize);

if (ptr[j] == NULL) {perror("malloc");}

printf("%10p\n", sbrk(0));}

printf("Program break is now: %10p\n", sbrk(0));

printf("Freeing blocks from %d to %d in steps of %d\n",freeMin,
freeMax, freeStep);

for (j = freeMin -1; j < freeMax; j += freeStep) {my_free(ptr[j]);}

printf("After my_free(), program break is: %10p\n", sbrk(0));

exit(EXIT_SUCCESS);}
```

OUTPUT



osd-190031187@team-osd:~

```
0x2487c90
0x248a4a8
0x248ccc0
0x248f4d8
0x2491cf0
0x2494508
0x2496d20
0x2499538
0x249bd50
0x249e568
0x24a0d80
0x24a3598
0x24a5db0
0x24a85c8
0x24aade0
0x24ad5f8
0x24afe10
0x24b2628
0x24b4e40
0x24b7658
0x24b9e70
0x24bc688
0x24beea0
0x24c16b8
0x24c3ed0
0x24c66e8
0x24c8f00
0x24cb718
0x24cdf30
0x24d0748
0x24d2f60
0x24d5778
0x24d7f90
0x24da7a8
0x24dcfc0
0x24df7d8
0x24e1ff0
0x24e4808
0x24e7020
0x24e9838
0x24ec050
0x24ee868
0x24f1080
0x24f3898
0x24f60b0
0x24f88c8
```

```
0x2c9b1e8
0x2c9da00
0x2ca0218
0x2ca2a30
0x2ca5248
0x2ca7a60
0x2caa278
0x2caca90
0x2caf2a8
0x2cb1ac0
0x2cb42d8
0x2cb6af0
0x2cb9308
0x2cbbb20
0x2cbe338
0x2cc0b50
0x2cc3368
0x2cc5b80
0x2cc8398
0x2ccabb0
0x2ccd3c8
0x2ccfbe0
0x2cd23f8
0x2cd4c10
0x2cd7428
0x2cd9c40
0x2cdc458
0x2cdec70
0x2ce1488
0x2ce3ca0
0x2ce64b8
0x2ce8cd0
0x2ceb4e8
0x2cedd00
0x2cf0518
0x2cf2d30
0x2cf5548
0x2cf7d60
0x2cfa578
0x2cfcd90
0x2cff5a8
0x2d01dc0
Program break is now:  0x2d01dc0
Freeing blocks from 1 to 999 in steps of 1
After my_free(), program break is:  0x2d01dc0
[osd-190031187@team-osd ~]$
```